Documentation for 3D printer mockup

This code is written as an interface to the server, designed to mimic the functionality of a 3D printer, used as a platform to solve some of the issues facing service invocation between SOA machines. It is designed, for the intents and purposes of this project, to act in a manner similar to a Status Enterprise Data Connector, but it is, in fact, a client application, and, as such, utilizes an available connection to the Status Enterprise Server.

Usage:

Upon starting the program, it will automatically connect if the program is compiled and run in debug mode, but if it is in release mode, the username and password must be provided as the first and second command line parameters

With the program running, commands should be entered into the Command field of the Printer001 asset on the Status Enterprise Server. The command syntax is as follows:

Single movements: The code is designed to take standard Reprap-flavor GCODE, but some functions are not implemented. They should not, however, cause the program to fail.

G1/G0 [X ###.###] [Y ###.###] [Z ###.###] [F ###.###] (extruder mockup not implemented)

G1 and G0 are processed identically, the parameters (except for G1/G0) can be given in any order, upper- and lowercase are allowed, white space after the letter is allowed, but not required, and any decimal or integer number can follow each letter. X, Y, and Z represent the 3 coordinates of the target destination, and F represents the travel speed. Any combination of the parameters may be used, but all are optional (making a call with no optional parameters will not crash, but produces no meaningful output)

G28

This is the homing function, and accepts no parameters; it resets the coordinates to <0,0,0>

Complex movements (print jobs): this is to specify a gcode file to print in its entirety

Print absolute\_path\_to\_gcode\_file.gcode

The word “print” is case-insensitive, and the command line accepts .gcode and .gco files. The path to the file should be absolute, and include the file extension. This system has not been adapted to take network locations as of yet, so it can only find files saved on the machine runn the printer mockup.

After entering the command into the Command field, the FuncCallMade variable needs to be flipped to true, indicating that a command is ready for the printer. Once this is done, the script should take care of the rest. Multiple jobs can be submitted at a time, and the printer will automatically queue them.

\*\*NOTE: the program possesses the capability to write a time to completion estimate to the server, and to update it in real time. HOWEVER, there is no functionality currently for actually estimating the time based on a gcode file. Instead, this has been implemented by taking a time estimate (most 3D print slicing softwares can provide this) and inserting it as a comment into the header of the .gcode file. The format for the comment is as follows:

; [print] time hh:mm[:ss]

The semicolon at the beginning serves to indicate that the text which follows is a comment, the word “print” is optional, both words are case insensitive, and the number of seconds is optional. Once the file is added to the print queue, the time will be immediately read, and added to the total estimated time displayed on the server.

Program flow explanation:

1. First, the connection to the server is established with the Connect() function.

If the server is hosted anywhere other than on the first machine it was created on (finoti), then the server information within Connect() must be edited.

1. Next, all of the NodeIds are initialized, establishing handles to all of the relevant values stored on the server.

If this were a real printer with a data connector, connecting the specific values from the printer to specific fields on the server would all be handled when setting up the data connector on the server.

1. A thread for taking input is started; this thread polls the server for command data, queues it for processing, and resets the input fields. It runs in the background for the entire duration of the program, but is automatically cleaned up when the program closes.
   1. As each command is queued, the time estimation is determined (with parseComments) and added to a separate queue (see 6), as well as to the current time estimation
2. A separate thread is created to write the time remaining in the queue to the server.

Both 3. and 4. would be unnecessary if this were a data connector instead of a client. This information-passing would be handled by the server.

1. Each command is dequeued and executed in sequence (with parseInput), updating the relevant server values (queue length, output field) as it goes
   1. parseInput serves as a rudimentary GCode interpreter, only processing movement commands and ignoring comments (denoted with “;”); every other command will return a “Command not recognized” message, but the program will not fail.
2. After each job is completed and dequeued, the time estimate is recalculated (since a 2 hour job may have been listed as a 2.5 hour job) based on a queue of the time estimations. As time passes, the current time remaining estimation counts down, but the values in the queue do not change, they are merely added together each time one is dequeued, and then that summation counts down as the job is processed.

To Do:

For large amounts of time (>1 day), the hour field in the Time Remaining field displays as a decimal, aka 1.01 for one day and one hour; this may be left as is, or it could be modified to display as a simple integer (25 in that case).

Some features of a 3D printer have not been fully implemented. This has served, and ought to successfully continue to serve, as a platform for solving the slightly higher-level challenges facing SOA systems, but for a more comprehensive system, extruder support and heating elements may be implemented in the mockup.

The X, Y, and Z limits have not been properly implemented. They are read from the server (since the machine is not real, they are arbitrary and may be changed every time the printer starts up), but they do not actually serve as limits; this functionality may be useful for sending error or warning messages to the server, but it may be possible to implement this via the server itself. If it becomes relevant to implement this in the printer software itself, the limits should be compared against the target destination with every parseInput() call (in the G1/G0 section) or in every Move() call.

The client currently parses all of the input from the server via intermittent polling of the input field; however, this could be replaced by a subscription (see *Status Enterprise User Guide (part 10) – Object Reference Model*), which seems like it would reduce the wasted processor cycles and wasted network traffic between the server and client (\*\*this merits further investigation, as establishing a subscription to a value on the server requires inputting a pollingTime, which indicates that there IS polling involved, but the subscription service seems to indicate that passing data is based on a notification method, rather than a polling method)

There should be a constant ‘heartbeat’ between the server and client, keeping the connection open by checking the input fields; if the polling is replaced, by a subscription (see above), it is *possible* that the connection will die over time, if there is no activity between the server and client. In this case, it would be necessary to ensure the previous connection is closed, and to reestablish a connection. This was attempted, however unsuccessfully, as the connections didn’t seem to close after the deadline passed, meaning all of the usable connections to the server ended up being monopolized by the printer mockup.

The output (and potentially input) should be more rigidly structured; the output, for example, ought to have a unique identifier to go with the message that the client can identify, like the client IP address or some similar ID, such that the client can access the information as it becomes available, without risking the data being overwritten and replaced with output from some subsequent job. Then, the output needs to either be listed as a block of text on the server (it was suggested that this be executed in XML format), or it should be assumed that the client has subscribed to the output value, and will receive notification when the value changes (the client would, in this case, wait until the data with the appropriate identifier is received)